PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 5 : G06F 9/46, H04M 3/50 — A1

(11) International Publication Number: **WO 94/20904**

(43) International Publication Date: 15 September 1994 (15.09.94)

(21) International Application Number: PCT/US94/00826

(22) International Filing Date: 24 January 1994 (24.01.94)

(30) Priority Data:
08/025,538 3 March 1993 (03.03.93) US

(71) Applicant: ROLM COMPANY [US/US]; 4900 Old Ironsides Drive, Santa Clara, CA 95054 (US).

(72) Inventors: KAMINSKY, Mark, E.; 177 Butano Avenue, Sunnyvale, CA 94086 (US). PERELMAN, Roberto; 1614 Eagle Drive, Sunnyvale, CA 94087 (US). BIPIN, Patel; 120 Plympton Court, San Jose, CA 95139 (US). ICHNOWSKI, Jeanne; 1051 Greenwood Avenue, Palo Alto, CA 94301 (US). YUAN, Chris; 823 Beaver Court, Fremont, CA 945390 (US).

(74) Agent: EINSCHLAG, Michael, B.; Siemens Corporation, Intellectual Property Dept., 186 Wood Avenue South, Iselin, NJ 08830 (US).

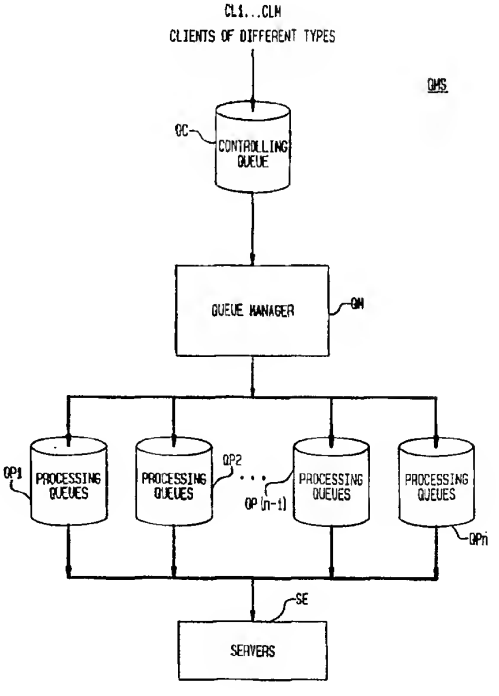
(81) Designated States: AU, CA, CZ, FI, HU, JP, RU, SK, European patent (AT, BE, CH, DE, DK, ES, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published
With international search report.

(54) Title: QUEUE MANAGING SYSTEM AND METHOD

(57) Abstract

In a queue management system for servicing of a number of clients representing different client types, a controlling queue queues clients in a predetermined order. A queue manager allocates and reallocates a number of processing queues, less than the number of client types, to match different one of said client types. The queue manager then places successive ones of the clients in the controlling queue into a processing queue matching the client type if there is a matching processing queue and allocates or reallocates an empty or emptied processing queue to the client type if there is no matching processing queue but there is an empty processing queue. A server empties the processing queues in batches. In the environment of a telephone system the clients are messages and the client types are codings in the messages for various destinations. The queue manager dedicates each of a number of processing queues to one of the destinations in the controlling queue, accesses the top message in the controlling queue, places the messages in a processing queue matching the destination code of the message if there is a match, and dedicates an empty processing queue to the target destination if there is no matching processing queue but there is an empty processing queue.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AT	Austria	**GB**	United Kingdom	**MR**	Mauritania
AU	Australia	**GE**	Georgia	**MW**	Malawi
BB	Barbados	**GN**	Guinea	**NE**	Niger
BE	Belgium	**GR**	Greece	**NL**	Netherlands
BF	Burkina Faso	**HU**	Hungary	**NO**	Norway
BG	Bulgaria	**IE**	Ireland	**NZ**	New Zealand
BJ	Benin	**IT**	Italy	**PL**	Poland
BR	Brazil	**JP**	Japan	**PT**	Portugal
BY	Belarus	**KE**	Kenya	**RO**	Romania
CA	Canada	**KG**	Kyrgystan	**RU**	Russian Federation
CF	Central African Republic	**KP**	Democratic People's Republic of Korea	**SD**	Sudan
CG	Congo			**SE**	Sweden
CH	Switzerland	**KR**	Republic of Korea	**SI**	Slovenia
CI	Côte d'Ivoire	**KZ**	Kazakhstan	**SK**	Slovakia
CM	Cameroon	**LI**	Liechtenstein	**SN**	Senegal
CN	China	**LK**	Sri Lanka	**TD**	Chad
CS	Czechoslovakia	**LU**	Luxembourg	**TG**	Togo
CZ	Czech Republic	**LV**	Latvia	**TJ**	Tajikistan
DE	Germany	**MC**	Monaco	**TT**	Trinidad and Tobago
DK	Denmark	**MD**	Republic of Moldova	**UA**	Ukraine
ES	Spain	**MG**	Madagascar	**US**	United States of America
FI	Finland	**ML**	Mali	**UZ**	Uzbekistan
FR	France	**MN**	Mongolia	**VN**	Viet Nam
GA	Gabon				

QUEUE MANAGING SYSTEM AND METHOD

Background of the Invention

This invention relates to methods and means for assigning a queue of clients requiring different types of services to servers offering the needed services. In one aspect, the invention relates more particularly to methods and means for managing queues of recorded telephone messages to permit transmission of batches of messages to intended destinations.

Such queue managing systems assign services to clients that have varying processing requirements depending on their types. For each type of client, such as A, B, C, etc., they provide a class of server, such as A, B, C, etc., that can service the client. They may instead furnish universal servers that can transform themselves to different classes of servers, e.g. A, B, C, etc. as the client needs.

Most queue managing systems service clients in a predetermined order, for example on a first-in first-out (FIFO) basis. They effect service by using one of two queuing schemes, namely using a single queue strategy or a multiple queue strategy.

In the single queue strategy, all clients wait in the same queue in the order that they arrive and then undergo servicing one at a time. If the top client in the queue is of the same type as the one currently being serviced, for example type A, then the server which is processing the type A client can process the next type A client immediately. Otherwise the system must change to a class of server appropriate to the next client, for example type B, or search the queue for the next client of the first type, type A.

In the environment of an existing telephone voice mail system, where clients take the form of recorded messages coded for various destinations, the single queue strategy works as follows. All messages wait in the same queue in the order of their arrival and a server then services and transmits each message one at a time. If messages targeted for the same destination follow each other, then the system can transmit them as a batch. Otherwise, the system changes the destination to that of the next message, or searches the queue for a message intended for the same destination as the first message and extracts several messages to the same destination together for transmission as a batch.

Such single queue systems are clearly inefficient. They cannot easily accommodate concurrent servicing of different clients or messages.

The multiple queue strategy utilizes multiple queues and associates each queue with a client type and a corresponding server type. Clients of the same type wait in the same queue in the order that they arrive. With a multiple queue scheme in the environment of a telephone voice mail system, the clients are recorded messages coded for various destinations. Each of the multiple queues contains messages for a specific destination. The messages for the same destination wait in the same queue in the order that they arrive. Such multiple queue arrangements require a number of queues equal to the maximum number of destinations which may be very large or unknown. Each message has to pass to the particular queue of the destination.

An object of this invention is to improve queue management systems, particularly for telephone systems using voice mail.

Another object of the invention is to overcome the aforementioned disadvantages.

Summary of the Invention

The invention involves assembling all the clients of varying types in a predetermined order in a controlling queue, periodically emptying each of a number of processing queues less than the number of client types, and transferring each successive client of a client type from the controlling queue only into a processing queue free of other client types. This allocates each processing queue to one client type until the processing queue is emptied and reallocates each emptied processing queue to the client type of the next client being transferred thereto.

Because the queue manager always appends, to each processing queue, the same type of client that processing queue already contains, the queue manager inherently allocates each processing queue to the type of client that the queue manger first places into that processing queue. This allocation continues until the server empties the processing queue. If the queue manager now places a new type of client in the empty processing queue the queue manager is deallocating the processing queue from the former type of client and reallocating it to the more recent type. This deallocation and reallocation allows a limited number of processing queues, such as fifty, to handle any number of client types, such as a thousand or more.

According to a specific aspect of the invention, namely one in a telephone environment, the clients are recorded voice mail messages coded for transmission to a number of destinations. All messages for all destinations appear in the controlling queue in a predetermined order, such as on a first-in first-out basis. The queue manager distributes the messages in the controlling queue, one by one, among a number of processing queues less than the number of destinations by placing each successive first message in the controlling

queue onto a processing queue whose destination matches the destination of the first message. If there is no matching processing queue, the queue manager places the first message into an empty processing queue if there is an empty processing queue. If there is no empty processing queue, the queue manager holds the message in the controlling queue until a transmission of messages from the processing queues empties a processing queue. The queue manager then reallocates that processing queue for transmission only to the new destination and places the next message into the empty processing queue.

The invention uses a limited number of queues to allow concurrent servicing of a large number of client types or destinations. There is virtually no limitation on the different client types or destinations. Clients of the same type or messages for the same destination wait in the same processing queue. The clients or messages wait in the same queue, normally in chronological order although different orderings are possible. Concurrent processing of different client types or message destinations is easily available. The system dynamically allocates queues to each client type or destination and deallocates them as they become empty and new client types or messages for new destinations arrive. The invention associates common attributes relevant to the processing of a given client type or destination with the corresponding queue for ease of processing.

Processing of multiple client types or messages for multiple destinations becomes more efficient than for a single queue scheme as well as for previous multiple queue systems. The invention allows for great flexibility in processing.

These and other features of the invention are pointed out in the claims forming a part of this specification. Other objects and advantages of the invention will become evident to those skilled in the art when read in light of the accompanying drawings.

Brief Description of the Drawings

Fig. 1 is a block diagram illustrating a queue management system embodying features of the invention.

Fig. 2 is a flow chart illustrating operation of the queue management system shown in Fig. 1 and embodying the invention.

Fig. 3 is a flow chart illustrating another method of operating the system shown in Fig. 1 and embodying the invention.

Fig. 4 is a flow chart illustrating the operation of the servers as they initiate servicing of the processing queues in Fig. 1.

Fig. 5 is yet another flow chart illustrating the operation of the servers in Fig. 1 as they conduct operation on the queues.

Fig. 6 is a block diagram of a telephone system incorporating an embodiment of the invention.

Fig. 7 is a block diagram of a queue management system embodying the invention in the environment of the telephone system in Fig. 6.

Fig. 8 is a flow chart illustrating steps in the operation of the queue management system shown in Figs. 6 and 7, and embodying the invention.

Fig. 9 is a flow chart illustrating further steps in the method of operating the system shown in Figs. 6 and 7 and embodying the invention.

Fig. 10 is a flow chart illustrating further steps in operating the system shown in Figs. 6 and 7 and embodying the invention.

Fig. 11 is yet another flow chart illustrating another operation of a queue processing controller shown in Figs 6 and 7 and embodying the invention.

Detailed Description of Preferred Embodiments

In the block diagram of a queue managing system QMS of Fig. 1, a queue manager QM manages a pool of queues, namely a controlling queue QC and a number of processing queues QP1, QP2, . . . QP(n-1), QPn. The controlling queue contains a sequence of clients CL1 to CLm of different client types TC1 to TCt requiring varying classes or types of processing. The number n of processing queues QP1 to QPn is less than the number t of the types TC1 to TCt of clients. The number n of processing queues may for example be 50 and the number t of types of clients may for example be 1000.

The controlling queue QC receives all clients of all client types, TC1 to TCt, and holds them in the order that they arrive. The queue manager QM removes the clients from the controlling queue in that same order. Hence, the controlling queue QC is a first in, first out, (FIFO) queue. However, this is only an example and the controlling queue may arrange the clients in another order as desired.

The queue manager QM places each top client in the controlling queue QC into a processing queue QP1 to QPn associated with that type of client. The queue manager establishes the association with the client type on the basis of the client type of the first client which the queue manager places into an empty processing queue QP1 to QPn. This occurs as follows.

Assuming at the outset that all processing queues QP1 to QPn are empty, and the first client in the controlling queue QC is a client of the type TC10, the queue manager places the client in the processing queue QP1. This step allocates or dedicates the processing queue QP1 to the client type TC10. The queue manager thereafter places only client types TC10 into the processing queue QP1 until one of the servers SE empties the processing queue QP1. If the next top client in the

controlling queue QC is a type TC143, the queue manager QM places the client of the type TC143 into the processing queue QP2 and hence dedicates or allocates the processing queue QP2 to the client type TC143.

If the next top client in the controlling queue QC is a type TC10, the queue manager places that client together with the other client TC10 into the processing queue QP1 rather than into one of the empty processing queues. Thereafter, the queue manager QM keeps placing the top client into the processing queue which the queue manager has allocated for that type of client, and if no such dedicated processing queue exists, the queue manager places the next top client into an empty processing queue. Each time an empty processing queue receives a client type, the queue manager dedicates that processing queue to that client type.

When the queue manager has dedicated or allocated all processing queues, and the next client being serviced is of a type to which no processing queue is dedicated, or allocated, the queue manager stops distributing clients from the controlling queue QC and waits for the servers SE to empty one of the processing queues QP1 to QPn. The servers SE service the processing queues QP1 to QPn individually either by removing the clients on a one by one basis or in batches. Such servicing may, for example, empty a processing queue and allow the queue manager to deallocate that processing queue and reallocate it to the type of client at the top of the controlling queue QC.

In an alternate embodiment, wherein clients are processed in batches with a maximum size, if the queue manager has dedicated or allocated all processing queues and the next client being serviced is of a type to which no processing queue has been dedicated or allocated, the queue manager examines the controlling queue, and transfers a client later in the controlling queue which matches a processing queue to its processing queue if

that transfer will not cause an extra batch to be processed.

The queue manager associates or maintains queue information for each processing queue. This queue information includes the current allocated client type and may contain other fields.

The invention is based on the recognition that although the maximum number of client types TC1 to TCt may be very large, at any one time there are usually only a limited number of clients actually awaiting servicing. Thus, by estimating how many client types exist in this subset of clients actually waiting for servicing, it is possible to allocate enough processing queues QP1 to QPn to hold them.

The queue manager QM does not permanently dedicate any processing queue to a fixed client type but only assigns the processing queues to one client type at a time. A processing queue QP1 to QPn is in use when the queue manager QM assigns it to a client type. Otherwise, the processing queue is empty and available. The processing queue is available for assignment or reassignment only when no client is in that processing queue.

Fig. 2 is a flow chart illustrating the operation of the queue management system QMS in Fig. 1. In step 100, the queue manager executes a wait to allow addition of clients to the controlling queue QC by one or more sources. Our processes start with a wait. This seems inefficient, but is actually not. The processes are cyclic, so after the initial wait, the processes proceed as if the wait were at the end. In most cases, when the system starts up, there are no clients, so if the process did not start with a wait, it would fall through to the wait without doing anything. Thus, starting with a wait is usually more efficient. Other implementations could differ. In step 104, the queue manager determines whether the controlling queue is

empty. If yes, the queue manager QM returns to wait at step 100. If the controlling queue QC is not empty but contains clients, the queue manager proceeds to step 107 and reads the client type of the first client. In step 110, the queue manager determines whether the client type of the first client matches the client type of any of the processing queues. That is, the queue manager determines from data it maintains whether it has allocated any of the processing queues to the client type found in step 107. In a sense, the queue manager QM is ascertaining whether any of the processing queues QP1 to QPn already contain client types of the type in step 107.

If the client type in step 107 matches a processing queue QP1 to QPn, queue manager QM proceeds to step 114 and removes the top, or first, client from the controlling queue and appends it to the processing queue QP1 to QPn allocated to the same client type. If the client type of the first client in step 107 does not match the client type in any of the processing queues in step 110, the queue manager proceeds to step 117 and asks whether there exist any available processing queues, namely a processing queue QP1 to QPn that is empty. If a processing queue QP1 to QPn is available, i.e., empty, the queue manager QM, in step 120, assigns the available new processing queue to that client type and removes the first client in step 107 and appends it to processing queue. It then returns to step 104 to ask whether the controlling queue QC is empty.

If no processing queue QP1 to QPn is empty, the processing queues may be effectively congested. In step 124, the queue manager QM ascertains whether the servers SE are servicing clients of at least one processing queue, or conversely if something is congesting or blocking all processing queues. If there exists no total block to servicing of the clients, that is, servicing of at least one processing queue is proceeding successfully, the queue manager QM returns the client to wait at step

100 and suspends execution of the entire process to allow time for servicing and emptying of the processing queue so it becomes available.

If the servers are not successfully servicing the clients in any of the processing queues, that is, if something is blocking or congesting servicing of all the processing queues, the queue manager goes to step 127. Here, the queue manager QM purges all of the processing queues QP1 to QPn by emptying the queues and returning the clients they contain to their sources. At this point, these clients are no longer part of the process although the sources may return the clients to the controlling queue. Other actions may be performed to correct the problems causing congestion. Once the queue manager QM has purged a processing queue QP1 to QPn, it can allocate that processing queue for the client type of the top client and transfer the top client in the controlling queue QC to the empty processing queue.

According to an embodiment of the invention, the queue manager QM purges only one or more of the processing queues QP1 to QPn in step 127. According to another embodiment of the invention, in step 127, the queue manager also examines the controlling queue, and returns any clients whose type was allocated to one of the congested processing queues. According to yet another embodiment of the invention, in step 127, instead of returning the clients to their sources, the queue manager returns the clients to the controlling queue. Two concerns must be met for this embodiment to work properly. First, any entries on the controlling queue (i.e., not yet in the processing queue) for a client type assigned to a processing queue must end up after the entries which are returned to the controlling queue from the processing queues, to maintain chronological order for clients of that type. Second, to avoid an infinite loop if there is a permanent problem, each client returned from the processing queue must have a return

count associated with it. A maximum return count would be assigned, and if the return count associated with the client reached that maximum value, the client would be returned to its source rather than to the controlling queue.

Another embodiment of the invention appears in the variation of Fig. 2 shown in the flow diagram of Fig. 3. Here, if the answer in step 124 is no, the processing queues are not all congested, the queue manager initiates a wait in step 130. This wait is comparable to the wait in step 100. Thereafter the queue manager returns directly to step 117. The operation in Fig. 3 is more efficient than returning to the step 100 in Fig. 2 if the controlling queue QC were strictly FIFO. In that case nothing could displace the top entry which would fail to match the client type of any processing queue QP1 to QPn. Hence, processing would always proceed through steps 104, 107, and 110 to step 117. In the embodiment of Fig. 3 the queue manager QM proceeds to step 117 directly. According to another embodiment of this invention, the source of a client is notified by the servers SE if the processing of that client fails.

According to another embodiment of the invention, the queue manager QM introduces other queue ordering in the controlling queues QC or QP1 to QPn, i.e., an ordering other than chronological. In one embodiment, the ordering is on the basis of priority. Another is a combination of priority and chronological. Any scalar field is available as priority. That is, the queue manager QM creates an order, either increasing or decreasing, by the value in that scalar field.

In the general case, a processing queue QP1 to QPn may contain many clients with many different priorities at a given time. The same situation may prevail in the controlling queue QC because all clients of all processing queues pass through the controlling queue first. According to one embodiment, the queue

manager QM uses a heap or linked list. According to another, it uses a FIFO queue with an auxiliary variable telling the highest priority value currently in the queue. These embodiments make a trade off between the setup time and the efficiency of finding the next client in the process.

The servers SE service the processing queues QP1 to QPn as shown in the flow charts of Figs. 4 and 5.

Fig. 4 illustrates the initiation of the processing and Fig. 5 illustrates the actual processing. The initiation of the processing and the processing take place separately so that the servers SE can process multiple queues simultaneously. The steps contain a "process started" check to insure that multiple processes are not taking place for the same queue.

Independent of the timing of the operation of the queue manager QM, the servers SE initiate a check of the processing queues QP1 to QPn in a predetermined sequence by first waiting in step 200. Our processes start with a wait. This seems inefficient, but is actually not. The processes are cyclic, so after the initial wait, the processes proceed as if the wait were at the end. In most cases, when the system starts up, there are no clients, so if the process did not start with a wait, it would fall through to the wait without doing anything. Thus, starting with a wait is usually more efficient. Other implementations could differ. In step 204 the servers SE check the next queue for clients ready for processing, and in step 207 ask whether the processing queue has any clients. If the answer is no, the servers SE go to step 210 and clean up or remove any queue information related to clients formerly in that queue and deallocate the queue. The servers then set up to check the next queue in step 214 and return to step 200.

If the answer in step 207 is yes, that the processing queue in question contains clients, the

servers go to step 220 to determine if that processing queue has already started and is receiving service. If yes, the servers progress to step 214 to set up to check the next queue. If the answer is no, the servers proceed to step 224 to mark the queue the subject of processing and to initiate servicing of the clients in the processing queue. The servers SE, in step 227, then determine if the servicing has successfully started. If not, the servers SE go back to step 200. If yes, the servers mark the queue as being processed in step 234 and go to step 214 before returning to step 200.

The processing steps after initiation appear in Fig. 5. Here, after a wait in step 260, the servers execute the process in step 264. Step 267 asks whether the process was successful. If yes, the servers remove the clients from the queue in step 270. In step 274 the servers determine if there are more clients in the queue being processed. If not, the servers continue to step 287 where the servers mark the queues as not processing, and the processing ends at that queue. If yes, the servers determine, in step 277, whether the maximum number of clients have been processed. Some servers handle clients in batches, which may or may not have a maximum size. If a server handles clients in batches, and that server handles the maximum number of clients in a batch, then the server will suspend processing until its next pass. If in step 277, the answer is yes, the servers have processed the maximum number. If not, the servers SE get the next client in step 280 and go back to step 264. If, in step 267, the answer is no, the servers go to step 284 to do error handling. According to one embodiment of the invention, error handling involves marking the processing queue for "retry" and specifying a retry time. The queue process controller, when initiating processing (Fig. 4), skips checking the queues marked for retry until the specified retry time.

According to an embodiment of the invention, the servers dispense with the error handling.

The block diagram of Fig. 6 illustrates an embodiment of the invention in the environment of a telephone system. Here, the clients CL1 to CLm of FIG. 2 are voice mail messages whose types vary with their destinations. In the telephone system of Fig. 6, a telephone branch exchange BE1 at a user location L1 connects a number of telephones T1, T2, . . . Tk to a public or private telephone network TN and to an associated voice mail system VM1. The telephone network TN connects the branch exchange BE1 to a number of other branch exchanges BE2 . . . BEq, all throughout locations LC1 to LCh possibly remote from the location L1. Each of the branch exchanges BE2 to BEq connects a number of telephones TL to the telephone network TN and to respective associated voice mail systems VM2 to VMq. Each voice mail system VM1 to VMq is of a known type which records messages arriving at its associated branch exchange BE1 to BEq from any of the telephones T1 - Tk and TL.

The locations LC1 to LCh of Fig. 6 may be widely spread. The location L1 of the branch exchange BE1 and voice mail system VM1 may for example be in Santa Clara, California, while the other locations may for example be in New York, New York; Boston, Massachusetts; and Nottingham, England, etc.

The voice mail system VM1 includes a queue management system QMT whose details appear in the diagram of Fig. 7. The voice mail system VM1 receives and stores all messages arriving from branch exchange BE1 either from telephones T1 to Tk at the local location L1 or telephones TL at remote locations LC1 to LCh. These include messages for retrieval by the telephones T1 to Tk and messages for transmission to remote telephones TL. Each message carries a destination code for a particular destination or particular number of destinations.

Typically, a destination is another voice mail system such as any one of VM2 to VMq. The queue management system QMT achieves its greatest efficiency when it collects a number of messages for transmission in a batch to a single destination such as a voice mail system. The coding of each message includes the number of the target voice mail system and the mail box number of the particular destination within the target voice mail system. The number of destinations may, for example be 1000.

In the voice mail system VM1, a selector (not shown) separates the recorded messages for transmission to remote telephones TL, i.e., at locations LC1 to LCh outside the branch exchange BE1 of the voice mail VM1, from the recorded messages intended for the local telephones T1 to Tk. The voice mail system VM1 allows each local telephone T1 to Tk to access messages targeted for those local telephones. The queue management system QMT in Fig. 7 receives, as clients, only messages ME1 to MEm coded for transmission to destinations DE1 to DEt at the remote telephones TL and the locations LC1 to LCh outside the branch exchange BE1.

Fig. 7 illustrates details of the queue management system QMT. The latter constitutes the system QMS of Fig. 1 in the context of a telephone system, and particularly a voice messaging system of Fig. 6. Fig. 7 is identical to Fig. 1 except that the clients CL1 to CLm take the specific form of messages ME1 to MEm, types TC1 to TCt take the specific form of destinations DE1 to DEt and the servers SE take the specific form of a queue process controller PC which dials the remote destination and transmits the messages. Like reference characters in Figs. 1 and 7 represent like parts.

In Fig. 7 the controlling queue QC receives messages ME1 to MEm for all destinations, DE1 to DEt, and holds them in the order that they arrive. The queue manager QM removes the messages from the controlling

queue QC in that same order. Hence, the controlling queue QC is a first in, first out, (FIFO) queue. However, this is only an example and the controlling queue may arrange the messages in another order as desired.

In Fig. 7, the queue manager QM places each top message of the controlling queue QC into a processing queue QP1 to QPn associated with one of the destinations. The queue manager QM establishes the association with the destination on the basis of the destination of the first message which the queue manager QM places into an empty processing queue. This occurs as follows.

Assuming at the outset that all processing queues QP1 to QPn are empty, and the first or top message in the controlling queue QC has a destination DE10, the queue manager QM places the message in the processing queue QP1. This step temporarily allocates or dedicates the processing queue QP1 to the destination DE10. The queue manager QM thereafter places only messages with destinations DE10 into the processing queue QP1 until the queue process controller PC empties the processing queue QP1. If the next top message in the controlling queue QC has a destination DE143, the queue manager QM places the message with the destination DE143 into the processing queue QP2 and hence dedicates or allocates the processing queue QP2 to the destination DE143.

If the next top message in the controlling queue QC has a destination DE10, the queue manager places that message after the other message with the destination DE10 into the processing queue QP1 rather than into one of the empty processing queues. Thereafter, the queue manager QM keeps placing the top message having a particular destination into the processing queue which the queue manager QM has allocated for that destination. If no such dedicated processing queue exists, the queue manager places the next top client into an empty processing queue. Each time an empty processing queue

receives a message with a particular destination, the queue manager dedicates that processing queue to that destination.

The queue process controller PC cycles through the processing queues at times independent of the action by the queue manager QM. At each processing queue, the queue process controller PC dials the number of the destination of the dedicated processing queue. If the call is completed, the queue process controller PC transmits a batch of the messages in that queue processor via the branch exchange. The maximum number of messages in a batch depends upon the number of messages that the system can transmit at any time which in turn depends upon the protocol that governs transmission of messages.

If the number of messages in the processing queue is less than the maximum in a batch such operation will empty the processing queue. This allows the queue manager QM to deallocate that processing queue and reallocate it to the destination of the message at the top of the controlling queue QC. However, if the number of messages in the processing queue exceeds the maximum of a batch, this procedure suspends the action until the next cycle of the queue process controller PC. The latter always goes on to check the next queue whether successful in sending a message or not.

When the queue manager has dedicated or allocated all processing queues, and the next message on the controlling queue is for a destination to which no processing queue is dedicated or allocated, the queue manager stops distributing messages from the controlling queue QC and waits for operation of the queue process controller PC and the branch exchange BE1 to empty and transmit the messages in one of the processing queues QP1 to QPn.

The queue manager QM and process controller PC associate or maintain queue information for each processing queue. This queue information includes the

current allocated destination code and may contain other fields, i.e., whether the processing queue is empty, being currently processed, the number of retries, the processing queue in retry, the next retry time, etc.

The invention is based on the recognition that although the maximum number of destinations DE1 to DEt may be very large such as 1000 or more, at any one time there are usually only a limited number of messages actually awaiting servicing and only a limited number of resources. Thus, by estimating how many destinations exist in this subset of messages actually waiting for transmission, it is possible to allocate enough processing queues QP1 to QPn such as fifty to hold them.

The queue manager does not permanently dedicate any processing queue to a fixed destination but only assigns the processing queues to one destination until the processing queue becomes empty. The queue manager then reallocates the empty queue. A processing queue QP1 to QPn is in use when the queue manager QM assigns it to a destination. Otherwise, the processing queue is empty and available. The processing queue is available for assignment or reassignment only when no message is in that processing queue.

Fig. 8 is a flow chart illustrating the operation of the queue management system QMT in Figs. 6 and 7. The operation is similar to that of Fig. 2 but in the telephone and voice messaging environment. In step 300, the queue manager QM executes a wait to allow addition of messages to the controlling queue QC. Our processes start with a wait. This seems inefficient, but is actually not. The processes are cyclic, so after the initial wait, the processes proceed as if the wait were at the end. In most cases, when the system starts up, there are no clients, so if the process did not start with a wait, it would fall through to the wait without doing anything. Thus, starting with a wait is usually more efficient. Other implementations could differ. In

step 304, the queue manager determines whether the controlling queue QC is empty. If yes, the queue manager QM returns to step 300. If the controlling queue QC is not empty but contains messages, the queue manager QM proceeds to step 307 and reads the destination code of the first message. In step 310, the queue manager determines whether the first destination code matches the destination code of any of the processing queues QP1 to QPn. That is, the queue manager QM checks the data it keeps regarding the processing queues to determine if it has allocated any of the processing queues to the destination found in step 307. In a sense, the queue manager QM is ascertaining whether any of the processing queues QP1 to QPn already contain messages to the destination in step 307.

If the destination in step 307 matches a processing queue QP1 to QPn, queue manager QM proceeds to step 314 and transfers the top, or first, message to the processing queue QP1 to QPn allocated to the same destination. If the destination of the first message in step 307 does not match the destination in any of the processing queues in step 310, the queue manager proceeds to step 317 and asks whether there exist any empty processing queues QP1 to QPn. If a processing queue QP1 to QPn is empty, the queue manager QM, in step 320, assigns the empty new processing queue to the destination of that message and transfers the top message in step 307 to that processing queue. It then returns to step 304 to ask whether the controlling queue QC is empty.

The lack of matching or empty processing queues QP1 to QPn may suggest congestion of the processing queues. In step 324, the queue manager ascertains from checking the results of the queue process controller PC if something is congesting or blocking delivery of messages from all the processing queues. If there is no total block to delivery, that is, delivery from at least one processing queue is proceeding successfully, the

queue manager QM leaves the message in the controlling queue to wait in step 300 and suspends execution of the entire process to allow time for a processing queue to empty. If something is blocking delivery of messages from all the processing queues, the queue manager QM goes to step 327. The queue manager QM purges one or more of the processing queues QP1 to QPn by emptying the queue and returning messages to their source. Other actions may be performed to correct the problems causing congestion.

This purging process returns the messages to their senders. Once the queue manager has purged a processing queue QP1 to QPn, it can transfer the top message from the controlling queue to the emptied processing queue, and thus allocate that processing queue to the destination of that message. According to another embodiment of the invention, the queue manager QM purges all the processing queues QP1 to QPn.

In Fig. 8, the queue manager QM can also introduce other queue ordering in the controlling queues QC or QP1 to QPn, i.e., an ordering other than chronological. Any scalar field is available as priority. That is, the queue manager QM creates an order, either increasing or decreasing, by the value in that scalar field.

In the general case, a processing queue QP1 to QPn may contain many messages with many different priorities at a given time. The same situation may prevail in the controlling queue QC because all messages of all processing queues pass through the controlling queue first.

The queue process controller PC directs the branch exchange BE1 to transmit the messages from the processing queues QP1 to QPn to their respective destinations as shown in the flow chart of Figs. 9 and 10. In Figs. 9 and 10, as in Figs. 4 and 5, the queue process controller PC divides the processing of the messages into two

separate tasks so as to make it possible to use multiple queues simultaneously. Fig. 9 illustrates the manner of initiating the transmission and Fig. 10, the manner of transmitting the messages of Fig. 9.

The steps of Fig. 9 are independent of the timing of the operation of the queue manager QM. Here, the queue process controller PC initiates a check of the processing queues QP1 to QPn in a predetermined sequence, one at a time and, between queue checks, waits as shown in step 400. Our processes start with a wait. This seems inefficient, but is actually not. The processes are cyclic, so after the initial wait, the processes proceed as if the wait were at the end. In most cases, when the system starts up, there are no clients, so if the process did not start with a wait, it would fall through to the wait without doing anything. Thus, starting with a wait is usually more efficient. Other implementations could differ. In step 404 the queue process controller PC checks the next queue for messages ready for transmission, and in step 407 asks whether the processing queue has any messages. If the answer is no, the queue process controller PC goes to step 410 and cleans up or removes any queue information related to messages formerly in that queue and deallocates the queue. The queue process controller PC then sets up to check the next queue in step 414 and returns to step 400.

If the answer in step 407 is yes, there are messages in the processing queue in question, the queue process controller PC goes to step 420 to determine if the branch exchange BE1 is already transmitting the messages from that processing queue. If yes, the queue process controller PC goes to step 414 to set up to check the next queue. In an embodiment using retries for error handling, step 414 may skip any queue in retry whose retry time has not been reached. If the answer is no, the queue process controller PC proceeds to step 424 to initiate the processing by marking the queue as the

subject of transmission and requesting the branch exchange BE1 to obtain a channel for delivering the message. The queue process controller PC, in step 427, determines if the channel was successfully obtained. If not, the queue process controller PC returns to step 400. If yes, the queue process controller PC marks the queue for processing in step 434 and returns to step 414 and then to step 400.

The processing steps appear in Fig. 10. Here, after a wait in step 460, the process controller PC in step 464 directs the branch exchange BE1 to dial the number of the destination of the messages in the processing queue and to transmit the first message of the processing queue through the channel it has made available. Our processes start with a wait. This seems inefficient, but is actually not. The processes are cyclic, so after the initial wait, the processes proceed as if the wait were at the end. In most cases, when the system starts up, there are no clients, so if the process did not start with a wait, it would fall through to the wait without doing anything. Thus, starting with a wait is usually more efficient. Other implementations could differ. In step 467, the process controller PC asks whether the transmission was successful. If yes, in step 470, queue process controller PC removes the message from the queue. In step 474, the process controller PC checks to see if there are any more messages in the processing queue. If the answer is yes, in step 477, the queue process controller asks whether the exchange BE1 has transmitted the maximum number of messages for each transmission. If not, in step 480, the queue process controller PC gets the next message and restarts the process in step 464.

Step 484 is optional and may be eliminated. Step 484 operates if in step 467 the branch exchange BE1 does not complete the call, i.e., the process is unsuccessful. The queue process controller PC updates

queue information associated with the queue to reflect the situation.

If message delivery in step 467 is not successful, several other embodiments of the invention perform error handling in step 484. On one embodiment, the queue process controller removes the message from the queue, returns it to the source, and ends servicing of the processing queue until the next cycle of the processing controller. In another embodiment, the queue process controller removes all the messages from the queue and returns them to their sources.

In another embodiment, the queue process controller leaves the message in the queue but sets a time to try again. The next attempt at message delivery from this queue will be considered a retry. The queue process controller PC keeps track of the number of retries. The queue manager makes note that the queue is in retry and monitors to be sure that not all queues are in retry when it checks for queue congestion at step 324 in Fig. 8.

If the message still does not go through after making retries, the queue process controller PC returns that message (and possibly all others to that destination) to its source.

When cycling through the queues to start a process, the queue process controller skips queues marked for retry until the time to process.

After completing error handling in step 484 the queue process controller PC then proceeds to step 487 to mark the queue as not processing and ends the process for that queue. It then waits for initiation of processing of the next queue by steps of Fig. 9.

If in step 474 the queue process controller PC determines that there are no more messages in the queue, or in step 477 determines that the exchange BE1 has processed the maximum number of messages, the controller PC proceeds to step 487.

If the call goes through successfully, the queue process controller PC delivers a batch of a predetermined maximum number of messages and the delivery ends. The predetermined number in the batch may not allow delivery of all messages in a processing queue. However, queue process controller PC can deliver more during its next cycle of checking the processing queues.

The process in Fig. 11 is a simplified version of that in Figs. 9 and 10. In this embodiment, processing of one queue must be completed before processing of another queue begins. Here, again, the steps of Fig. 11 are independent of the timing of the operation of the queue manager QM. Here, the queue process controller PC initiates a check of the processing queues QP1 to QPn in a predetermined sequence, one at a time and, between queue checks, waits as shown in step 500. Our processes start with a wait. This seems inefficient, but is actually not. The processes are cyclic, so after the initial wait, the processes proceed as if the wait were at the end. In most cases, when the system starts up, there are no clients, so if the process did not start with a wait, it would fall through to the wait without doing anything. Thus, starting with a wait is usually more efficient. Other implementations could differ. In step 504 the queue process controller PC checks the next queue for messages ready for transmission, and in step 507 asks whether the processing queue has any messages. If the answer is no, the queue process controller PC goes to step 510 and cleans up or removes any queue information related to messages formerly in that queue and deallocates the queue. The queue process controller PC then sets up to check the next queue in step 514 and returns to step 500.

If the answer in step 507 is yes, there are messages in the processing queue, the queue process controller PC directs the branch exchange BE1 to dial the number of the destination of the messages in the

processing queue and to transmit the first message of the processing queue through the channel it has made available. In step 567, the process controller PC asks whether the transmission was successful. If yes, in step 570, queue process controller PC removes the message from the queue. In step 574, the process controller PC checks to see if there are any more messages in the processing queue. If the answer is yes, in step 577, the queue process controller asks whether the exchange BE1 has transmitted the maximum number of messages for each transmission. If not, in step 580, the queue process controller PC gets the next message and restarts the process in step 520.

If step 574 determines that there are no more messages in the queue, or step 577 indicates that the exchange BE1 has processed the maximum number of messages, the controller PC proceeds to step 514.

Step 584 is optional and may be eliminated. If in step 567, the branch exchange BE1 does not complete the call, i.e., the process is unsuccessful, step 584 sets up queue information to reflect the situation. In step 584, the next attempt at message delivery is a retry. If the message still does not go through after making retries, the queue process controller PC returns that message and possibly all others to that destination.

If the call goes through successfully, the queue process controller PC delivers a batch of a predetermined maximum number of messages and the delivery ends. The predetermined number in the batch may not allow delivery of all messages in a processing queue. However, queue process controller PC can deliver more during its next cycle of checking the processing queues.

The invention has the advantage of providing efficient service even though the maximum number of destinations may be very large such as one thousand. The invention is based on the recognition that at any one time there are usually only a limited number of

destinations actually waiting to be serviced. Thus, only a small number of processing queues QP1 to QPn such as 50 can service a large number of destinations. The controlling queue serves as an overflow queue if the number of processing queues is insufficient.

The invention offers the advantage of preventing the messages for the busiest destinations from monopolizing the processing queues. This arises because each message must wait its turn in the controlling queue QC.

According to the invention, the queue manager does not permanently dedicate processing queues QP1 to QPn to fixed destinations but dedicates them temporarily to a particular destination. One of the processing queues QP1 to QPn is in use when dedicated to one destination. Once emptied, the processing queue becomes available for allocation to another destination. Also, the queue manager may add messages to a processing queue while processing is taking place.

According to another feature of the invention, the process controller PC receives other information to determine when to trigger the activities of a processing queue. For example, delivery time and delivery count determines when and if the delivery process should make an attempt to send a message to a destination, how many attempts have been made, and if another retry is pending.

According to another embodiment of the invention, the controlling queue QC uses ordering methods other than chronological for the queues. In one instance, the control QC places the messages on a priority basis or a combination of priority and first-in first-out basis. The controlling queue QC can use any scalar field as a priority and process messages in order of increasing or decreasing value in that field. According to still another embodiment of the invention, the controlling queue QC orders many different priorities in one of the processing queues QP1 to QPn at a given time. For

efficiency, a heap or linked list may replace the first-in first-out queue. Another scheme uses the first-in first-out queue with an auxiliary variable telling the highest priority value currently in the queue. This results in a trade off between the set up time and the efficiency of finding the next message to process. According to another embodiment of the invention, the controlling queue manager QM uses different processing queues for clients of the same type but different priorities.

According to an embodiment of the invention the queue manager QM handles congestion in the processing queues QP1 to QPn as follows. The queue manager QM suspends any message movement out of the controlling queue QC until at least one of the processing queues QP1 to QPn becomes available. Alternately, the queue manager QM purges any of processing queues QP1 to QPn that has a processing problem based upon the queue information associated with each processing queue. This makes some of the processing queues QP1 to QPn available during congestion and speeds the overall processing.

Yet another method of treating congestion involves waiting until all the processing queues QP1 to QPn have processing problems and purge all the processing queues QP1 to QPn at once. This makes all of the processing queues QP1 to QPn available.

In the telephone or voice messaging environment, the invention assigns messages only to processing queues which are free of messages targeted for other destinations. Thus messages for the same destination go to the same processing queue even though there can exist a virtually unlimited number of destinations and only a limited number of queues. The sources put messages in the controlling queue QC first and the queue manager QM moves them to one of the processing queues for further processing. This isolates knowledge of the processing queues QP1 to QPn to the

queue manager QM and the process controller PC. External processes which send messages to the controlling queue know only the controlling queue QC.

The invention allows concurrent processing of messages with different destinations and keeps a chronological order for those messages with the same message destination. The invention dynamically uses a queue for holding messages of the same message type, e.g. destination. One of the processing queues QP1 to QPn can hold messages for one destination at one time and messages for another destination at another time. The queue manager QM knows that the processing queue is either in use, in a retry condition, or empty and available.

The invention permits ordering messages by priority in addition to or instead of by chronology. The invention uses a limited number of queues to allow concurrent servicing of large numbers of destinations. There is in fact no limitation on the different destinations. Messages for the same destination wait in the same processing queue. The messages in the same processing queue are normally in chronological order unless an alternative approach is desired. Concurrent processing of different messages is easily available. The system dynamically allocates processing queues QP1 to QPn to each destination and deallocates them from such destination. The invention associates common attributes relevant to the processing of a given message with the corresponding queue for ease of processing.

Processing of multiple messages for the same destination becomes more efficient than for a single queue system as well as for previous multiple queue systems. The invention allows for great flexibility in processing.

If message delivery is not successful, several other embodiments of the invention perform error handling. On one embodiment, the queue process

controller or the queue manager removes the message from the queue, returns it to the source, and ends servicing of the processing until the next cycle of the processing controller. In another embodiment, the queue process controller or the queue manager removes all the messages. In another embodiment, the queue process controller or the queue manager leaves the message in the queue but sets a time to try again. When cycling through the queues to start a process, the queue process controller skips the queue until the time to process.

While embodiments of the invention have been described in detail, it will be evident to those skilled in the art that the invention may be embodied otherwise without departing from its spirit and scope. For example, although the queue manager has been described in terms of a voice mail message system, those or ordinary skill in the art should readily appreciate that the method and apparatus described above also apply to all other types of store-and-forward messaging systems. For example, embodiments of the present invention apply as well to store-and-forward messaging systems for, for example, facsimile messages (FAX), electronic mail messages (E-mail), video messages, as well as multimedia messages of all sorts.

What is claimed is:

1. A queue management system for a number of clients in a system, the clients representing a plurality of client types, the queue management system comprising:

a controlling queue for queuing clients;

a plurality of processing queues less than the number of client types for receiving clients and each for being periodically emptied of clients; and

a queue manager for transferring clients of a client type from said controlling queue into a processing queue free of clients of other client types, whereby the processing queue is allocated to the one client type until the processing queue is emptied.

2. A system as in claim 1, wherein queue manager further comprises means for retaining a client having a client type in the controlling queue if there is no empty processing queue and if there is no processing queue with one or more clients having the client type.

3. A system as in claim 1, wherein said queue manager includes:

means for determining the client type of a client in the controlling queue;

means for determining if there is a processing queue having a client with the client type; and

means for transferring a client to a matching processing queue, that is, a processing queue that includes a client having the client type.

4. A system as in claim 3, wherein said queue manager further includes:

means for searching for an empty processing queue if there is no matching processing queue; and

means for transferring the client to an empty processing queue if there is an empty processing queue.

5. A system as in claim 4, wherein said queue manager further includes:

means for ascertaining, when there is no empty processing queue, whether any processing queue is being emptied; and

means, if no processing queue is being emptied, for emptying at least one of the processing queues and for transferring a client from the controlling queue to one of the emptied processing queues.

6. A system as in claim 5, wherein said queue manager further includes:

means for suspending the process of removing clients from the controlling queue if all processing queues have been allocated and processing queues are being emptied.

7. A system as in claim 1, further comprising:

processing means for processing clients in each of said processing queues in turn as a batch;

means for determining the maximum number of clients allowed in a batch; and

means for determining when that maximum number has been reached.

8. A system as in claim 7, wherein said processing means includes:

means for checking a processing queue for clients;

means for checking the next processing queue for clients if there are no clients in the processing queue being checked;

means for examining the processing queue being checked to see if that processing queue is being processed; and

means for starting the processing if the processing queue is not being processed.

9. A queue management method for servicing of a number of clients in a system, the clients representing a plurality of client types, the queue management method comprising the steps of:

transferring the clients to a controlling queue;

periodically emptying each of a plurality of processing queues less than the number of client types; and

transferring clients of a client type from said controlling queue into a processing queue free of other client types, whereby the processing queue is allocated to the one client type until the processing queue is emptied.

10. A method as in claim 9, further comprising the step of retaining a client having a client type in the controlling queue if there is no empty processing queue and if there is no processing queue with one or more clients having the client type.

11. A method as in claim 9, wherein said transferring step includes:

determining the client type of a client in the controlling queue;

determining if there is a processing queue having a client with the client type; and

transferring a client to a matching processing queue, that is, a processing queue that includes a client having the client type.

12. A method as in claim 11, wherein said transferring step further includes the steps of:

searching for an empty processing queue if there is no matching processing queue; and

transferring the client to an empty processing queue if there is an empty processing queue.

13. A method as in claim 12, wherein said transferring step further includes the step of:

ascertaining, when there is no empty processing queue, whether any processing queue is being emptied; and

if no processing queue is being emptied, emptying at least one of the processing queues and

transferring a client from the controlling queue to one of the emptied processing queues.

14. A method as in claim 13, wherein said transferring step further includes:

suspending the process of removing clients from the controlling queue if all processing queues have been allocated and processing queues are being emptied.

15. A method as in claim 9, further comprising the steps of:

processing clients in each of said processing queues in turn as a batch;

determining the maximum number of clients allowed in a batch; and

determining when that maximum number has been reached.

16. A method as in claim 15, wherein said processing step includes:

checking a processing queue for clients;

checking the next processing queue for clients if there are no clients in the processing queue being checked;

examining the processing queue being checked to see if that processing queue is being processed; and

starting the processing if the processing queue is not being processed.

17. A store-and-forward messaging system comprising:

means for receiving messages having destination codes for transmission to one of a number of target destinations;

a controlling queue for queuing the messages;

a plurality of processing queues less than the number of target destinations; and

a queue manager for transferring a message for a destination code from said controlling queue into a processing queue free of messages to other destination codes, whereby the processing queue is allocated to the

one destination code until the processing queue is emptied.

18. A system as in claim 17, wherein said queue manager further comprises means for retaining a message having a destination code in the controlling queue if there is no empty processing queue and if there is no processing queue with one or more clients having the destination code.

19. A system as in claim 17, wherein said queue manager includes:

means for determining the destination code of a message in the controlling queue;

means for determining if there is a processing queue having a message with the destination code; and

means for transferring a message to a matching queue, that is, a processing queue that includes a message having the destination code.

20. A system as in claim 19, wherein said queue manager further includes:

means for searching for an empty processing queue if there is no matching processing queue; and

means for transferring the message to an empty processing queue if there is an empty processing queue.

21. A system as in claim 20, wherein said queue manager further includes:

means for ascertaining, when there is no empty processing queue, whether any processing queue is being emptied; and

means, if no processing queue is being emptied, for emptying at least one of the processing queues and for transferring a message from the controlling queue to one of the emptied processing queues.

22. A system as in claim 21, wherein said queue manager further includes:

means suspending the process of removing messages from the controlling queue if all processing

queues have been allocated and processing queues are being emptied.

23. A system as in claim 17, further comprising:

processing means for processing messages in each of said processing queues in turn as a batch;

means for determining the maximum number of messages allowed in a batch; and

means for determining when that maximum number has been reached.

24. A system as in claim 23, wherein said processing means includes:

means for checking a processing queue for messages;

means for checking the next processing queue for messages if there are no messages in the processing queue being checked;

means for examining the processing queue being checked to see if that processing queue is being processed; and

means for starting the processing if the processing queue is not being processed.

25. A queuing method for transmission of messages having destination codes to one of a number of specific target destinations, the method comprising the steps of:

transferring the messages to a controlling queue;

periodically emptying each of a plurality of processing queues less than the number of destinations; and

transferring messages for a destination from said controlling queue into a processing queue free of other destinations, whereby the processing queue is allocated to the one destination until the processing queue is emptied.

26. A method as in claim 25, further comprising the step of retaining a message having a destination in the controlling queue if there is no empty processing queue and if there is no processing queue with one or more messages having the destination.

27. The system of claim 5 wherein the means for emptying at least one of the processing queues further comprises means for returning a client from an emptied processing queue to the controlling queue.

28. The system of claim 4 wherein the queue manager further comprises:

means for ascertaining, when there is no empty processing queue, whether any processing is being emptied; and

means, if no processing queue is being emptied, for emptying at least one of the processing queues and for removing clients having the client type of a client on an emptied processing queue from the controlling queue.

FIG. 1

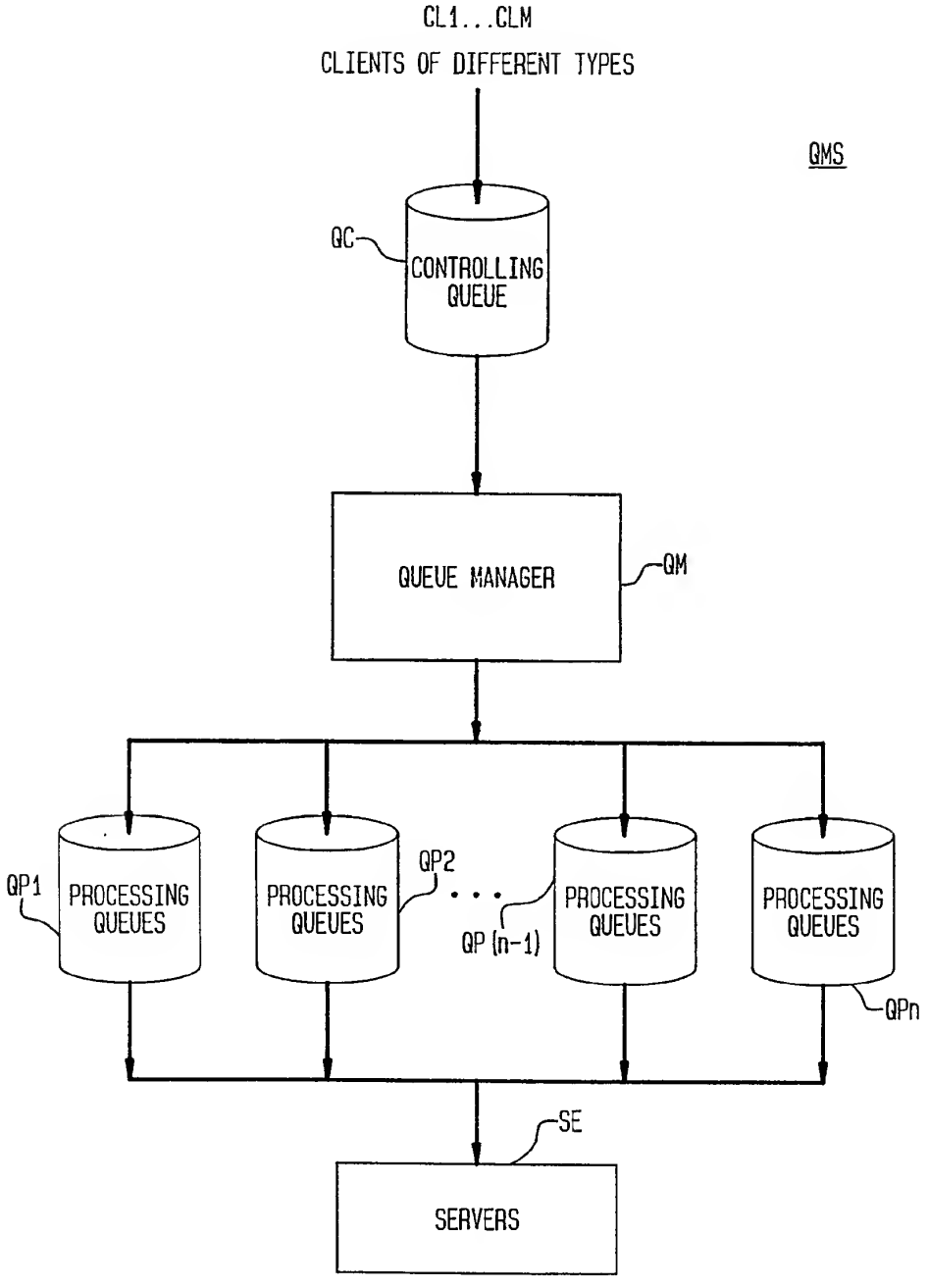

CL1...CLM
CLIENTS OF DIFFERENT TYPES
QMS
QC
CONTROLLING QUEUE
QUEUE MANAGER
QM
QP1
PROCESSING QUEUES
PROCESSING QUEUES
QP2
QP (n-1)
PROCESSING QUEUES
PROCESSING QUEUES
QPn
SE
SERVERS

FIG. 2

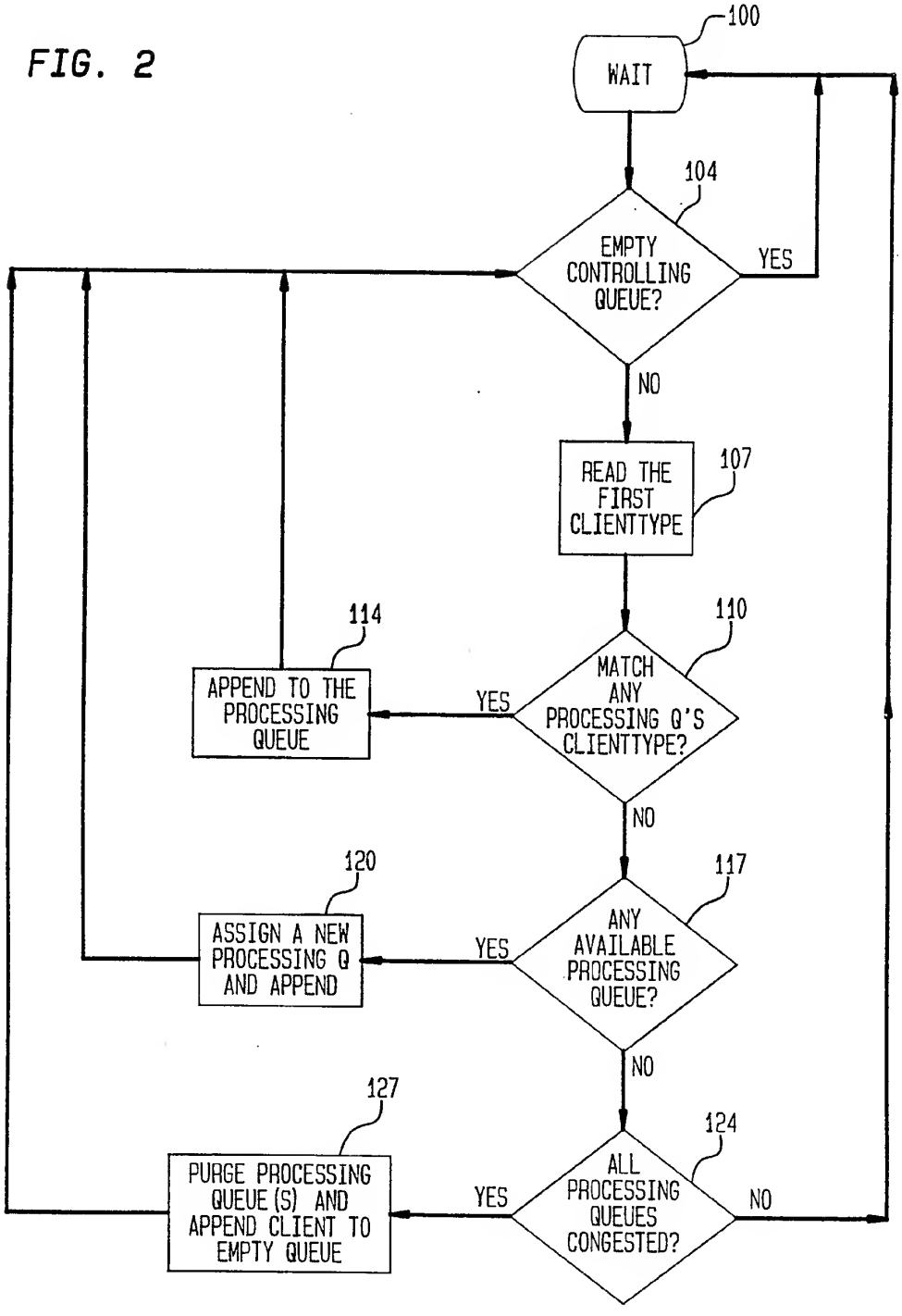

100
WAIT
104
EMPTY CONTROLLING QUEUE?
YES
NO
READ THE FIRST CLIENTTYPE
107
110
MATCH ANY PROCESSING Q'S CLIENTTYPE?
YES
114
APPEND TO THE PROCESSING QUEUE
NO
117
ANY AVAILABLE PROCESSING QUEUE?
YES
120
ASSIGN A NEW PROCESSING Q AND APPEND
NO
124
ALL PROCESSING QUEUES CONGESTED?
YES
NO
127
PURGE PROCESSING QUEUE(S) AND APPEND CLIENT TO EMPTY QUEUE

FIG. 3

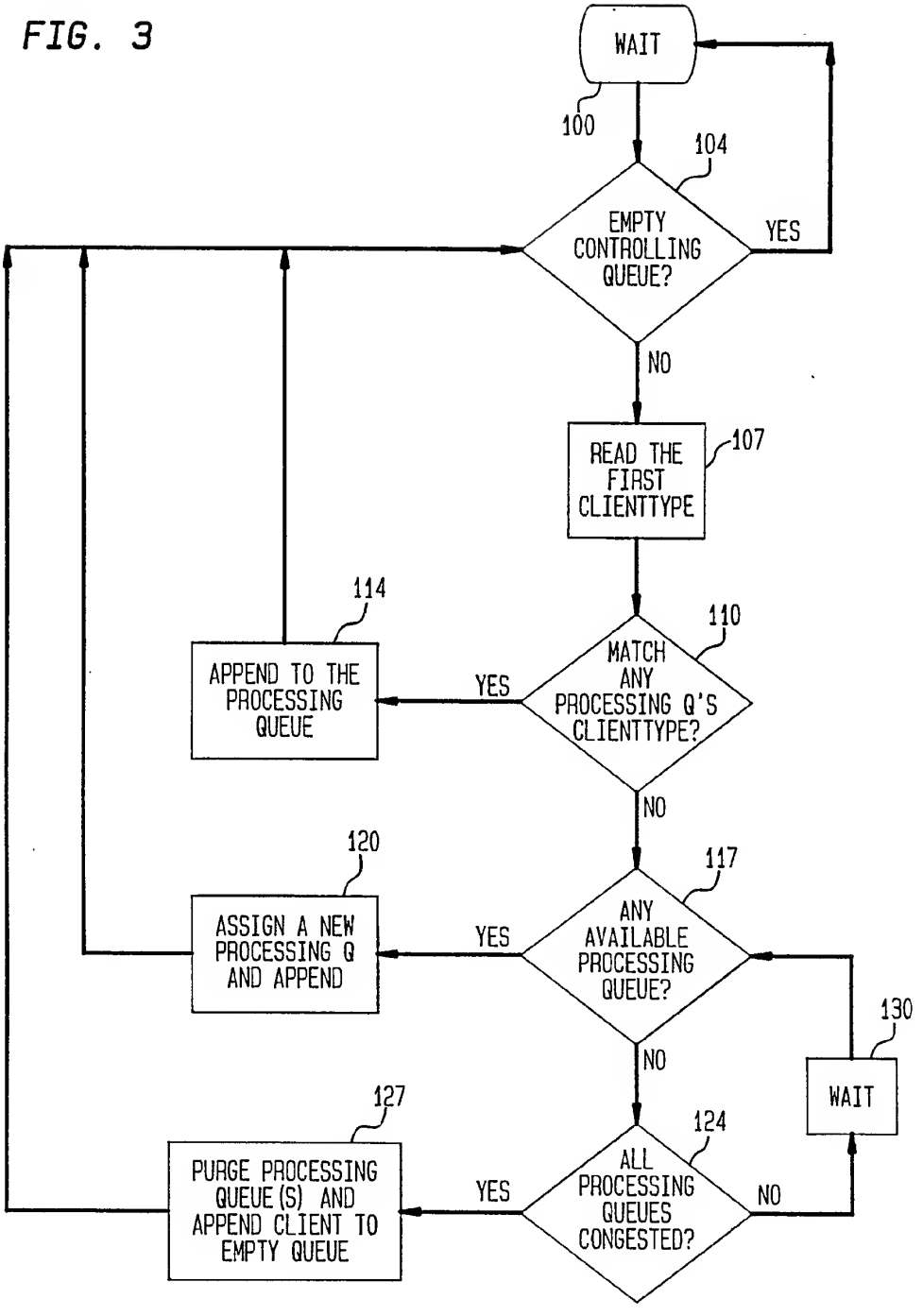

WAIT
100
104
EMPTY CONTROLLING QUEUE?
YES
NO
READ THE FIRST CLIENTTYPE
107
114
110
APPEND TO THE PROCESSING QUEUE
YES
MATCH ANY PROCESSING Q'S CLIENTTYPE?
NO
120
117
ASSIGN A NEW PROCESSING Q AND APPEND
YES
ANY AVAILABLE PROCESSING QUEUE?
130
NO
WAIT
127
124
PURGE PROCESSING QUEUE(S) AND APPEND CLIENT TO EMPTY QUEUE
YES
ALL PROCESSING QUEUES CONGESTED?
NO

FIG. 4

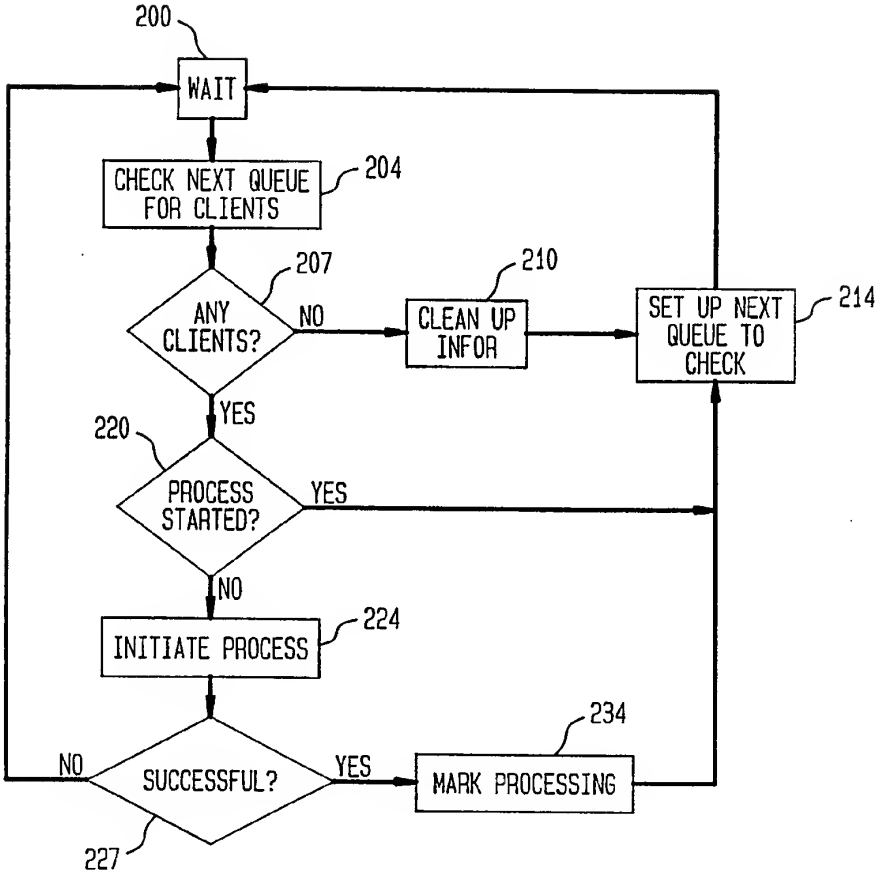

200
WAIT
CHECK NEXT QUEUE FOR CLIENTS
204
207
ANY CLIENTS?
NO
210
CLEAN UP INFOR
SET UP NEXT QUEUE TO CHECK
214
YES
220
PROCESS STARTED?
YES
NO
INITIATE PROCESS
224
234
NO
SUCCESSFUL?
YES
MARK PROCESSING
227

FIG. 5

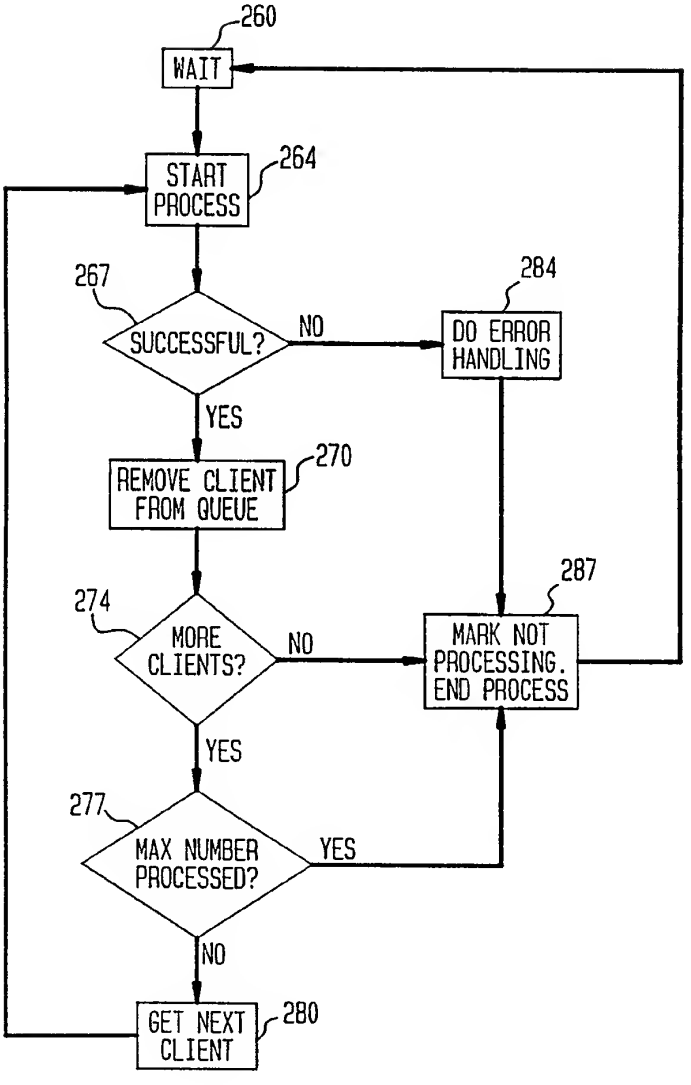

260
WAIT
START PROCESS
264
267
SUCCESSFUL?
NO
284
DO ERROR HANDLING
YES
REMOVE CLIENT FROM QUEUE
270
274
MORE CLIENTS?
NO
287
MARK NOT PROCESSING. END PROCESS
YES
277
MAX NUMBER PROCESSED?
YES
NO
GET NEXT CLIENT
280

FIG. 6

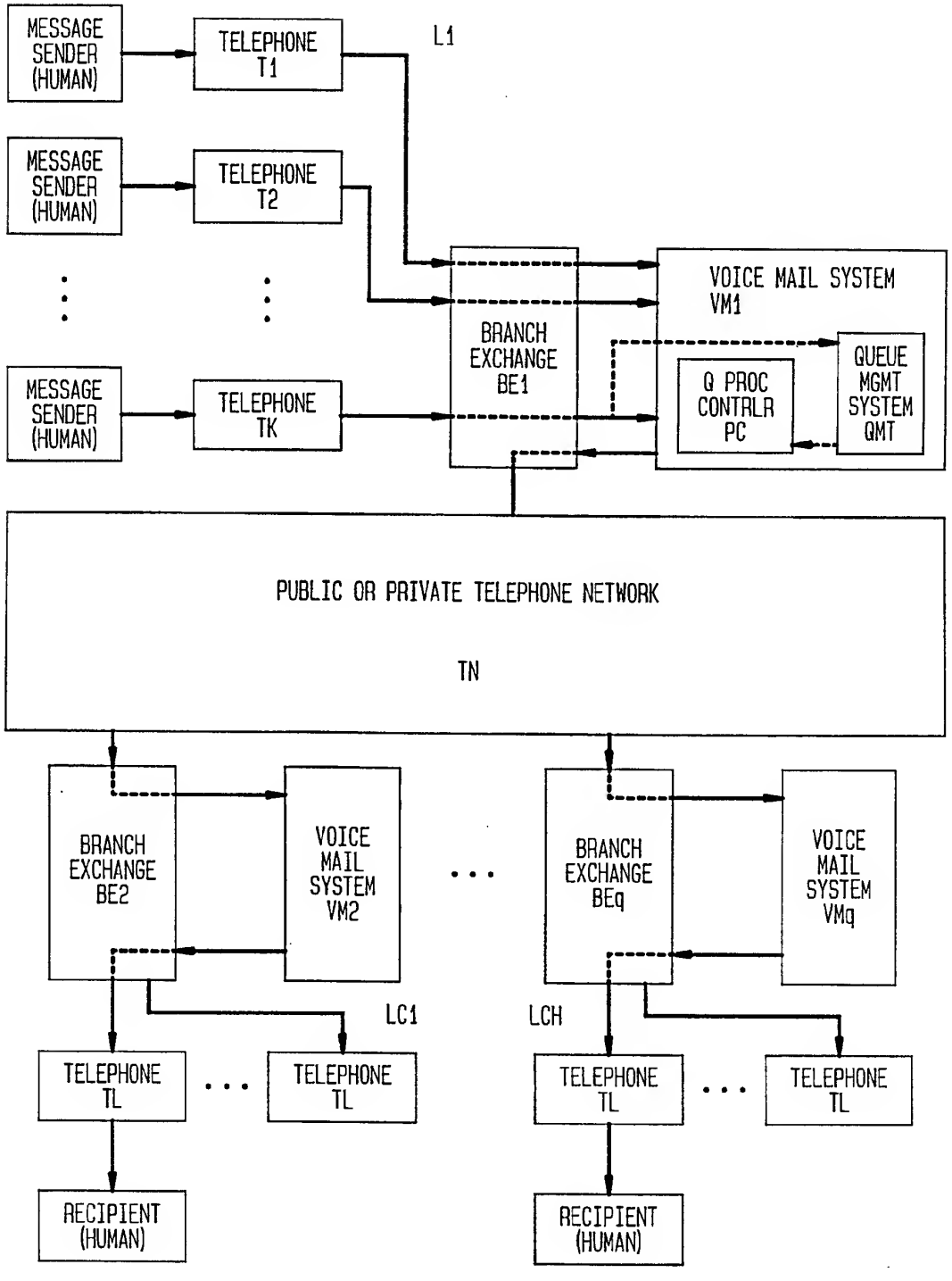

MESSAGE SENDER (HUMAN)
TELEPHONE T1
L1
MESSAGE SENDER (HUMAN)
TELEPHONE T2
MESSAGE SENDER (HUMAN)
TELEPHONE TK
BRANCH EXCHANGE BE1
VOICE MAIL SYSTEM VM1
Q PROC CONTRLR PC
QUEUE MGMT SYSTEM QMT
PUBLIC OR PRIVATE TELEPHONE NETWORK
TN
BRANCH EXCHANGE BE2
VOICE MAIL SYSTEM VM2
BRANCH EXCHANGE BEq
VOICE MAIL SYSTEM VMq
LC1
LCH
TELEPHONE TL
TELEPHONE TL
TELEPHONE TL
TELEPHONE TL
RECIPIENT (HUMAN)
RECIPIENT (HUMAN)

FIG. 7

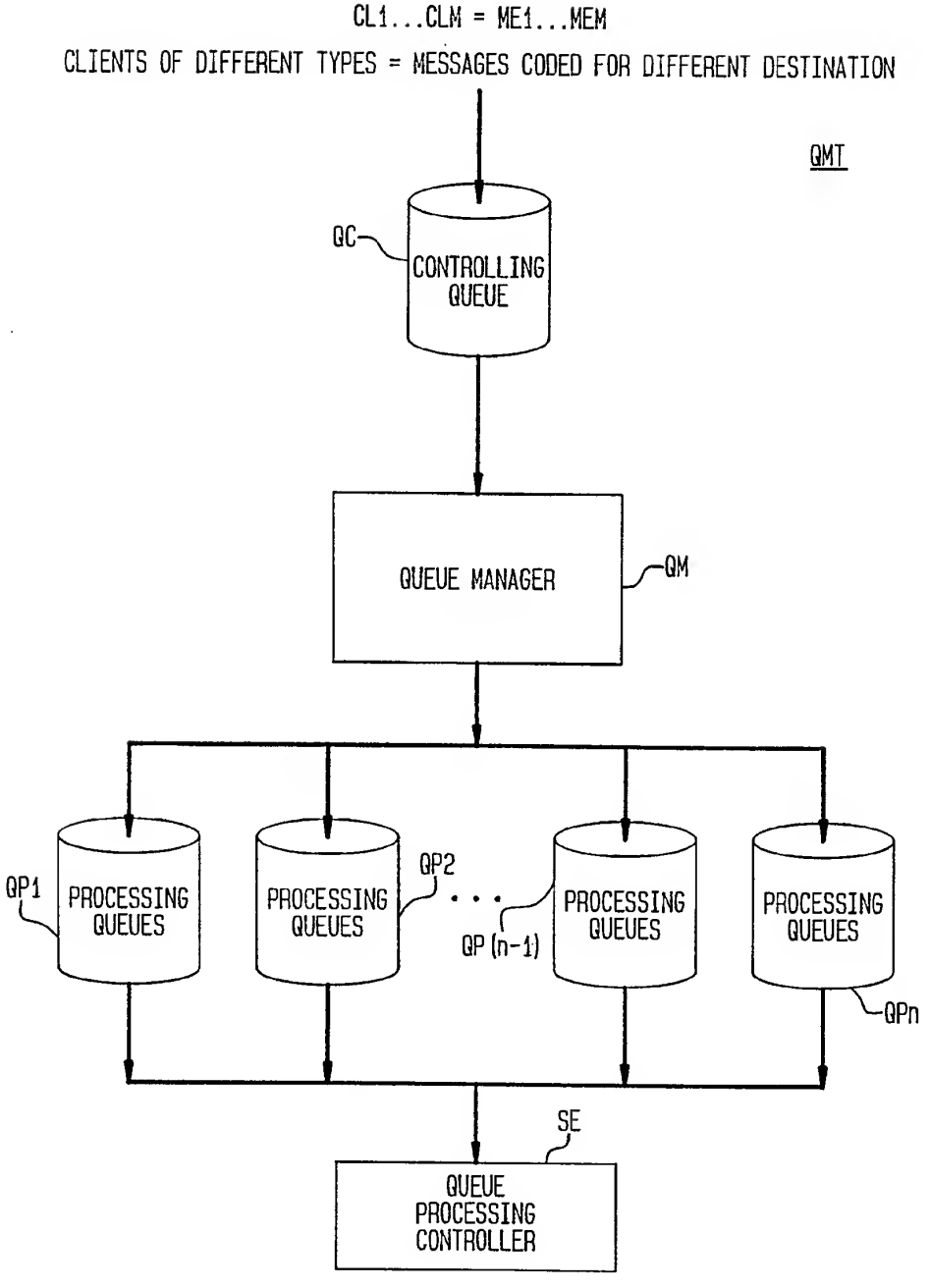

CL1...CLM = ME1...MEM
CLIENTS OF DIFFERENT TYPES = MESSAGES CODED FOR DIFFERENT DESTINATION
QMT
QC
CONTROLLING QUEUE
QUEUE MANAGER
QM
QP1
PROCESSING QUEUES
PROCESSING QUEUES
QP2
QP (n-1)
PROCESSING QUEUES
PROCESSING QUEUES
QPn
SE
QUEUE PROCESSING CONTROLLER

FIG. 8

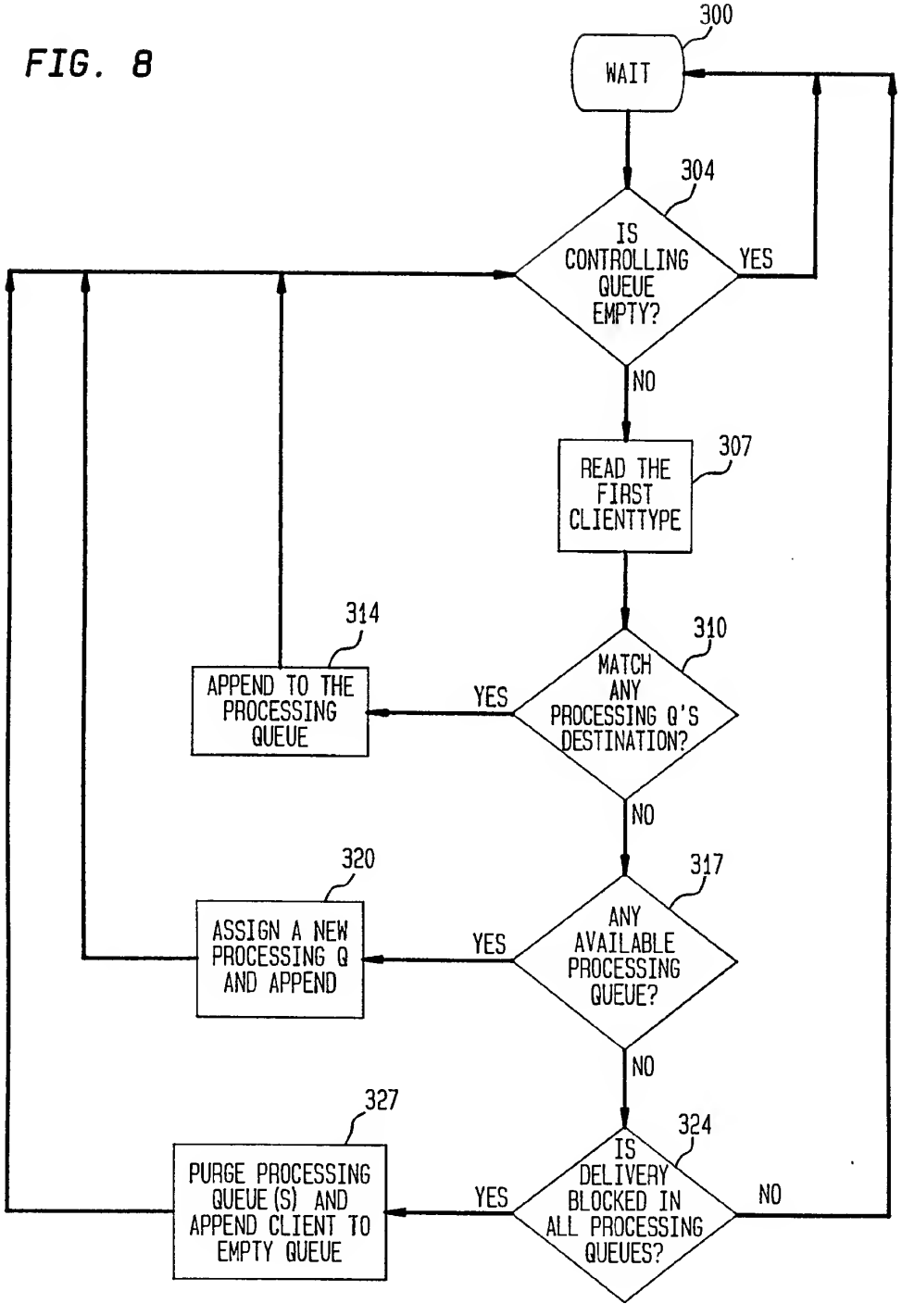

300
WAIT
304
IS CONTROLLING QUEUE EMPTY?
YES
NO
307
READ THE FIRST CLIENTTYPE
310
MATCH ANY PROCESSING Q'S DESTINATION?
YES
314
APPEND TO THE PROCESSING QUEUE
NO
317
ANY AVAILABLE PROCESSING QUEUE?
YES
320
ASSIGN A NEW PROCESSING Q AND APPEND
NO
324
IS DELIVERY BLOCKED IN ALL PROCESSING QUEUES?
YES
NO
327
PURGE PROCESSING QUEUE(S) AND APPEND CLIENT TO EMPTY QUEUE

FIG. 9

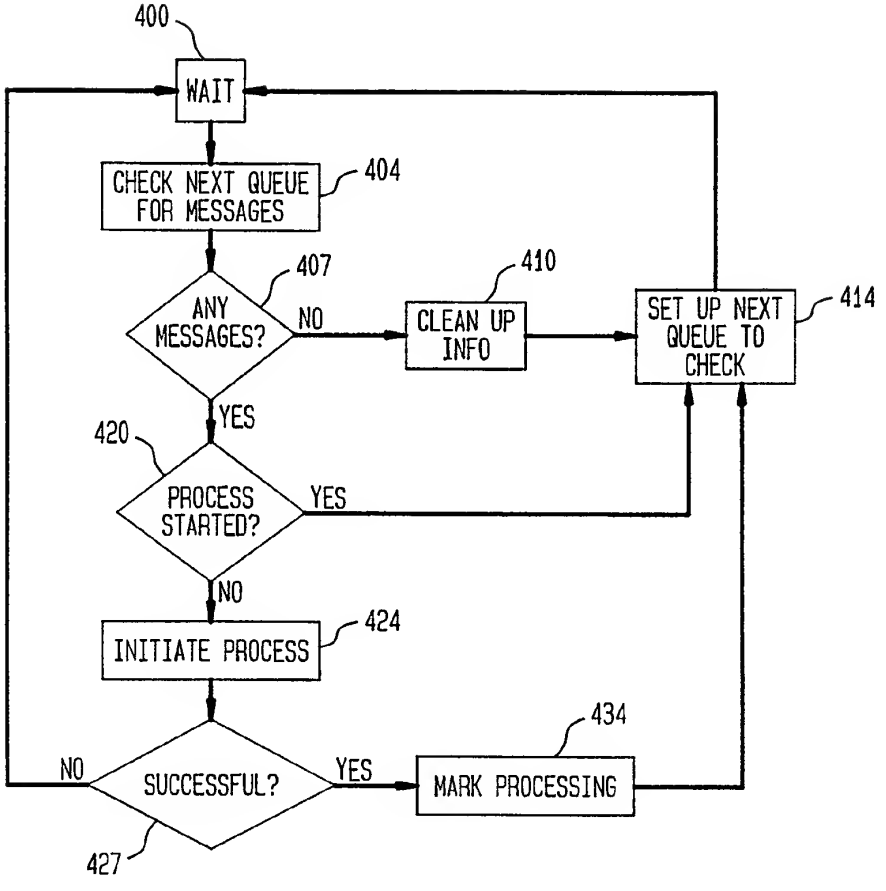

400
WAIT
CHECK NEXT QUEUE FOR MESSAGES
404
ANY MESSAGES?
407
NO
CLEAN UP INFO
410
SET UP NEXT QUEUE TO CHECK
414
YES
420
PROCESS STARTED?
YES
NO
INITIATE PROCESS
424
SUCCESSFUL?
NO
YES
MARK PROCESSING
434
427

FIG. 10

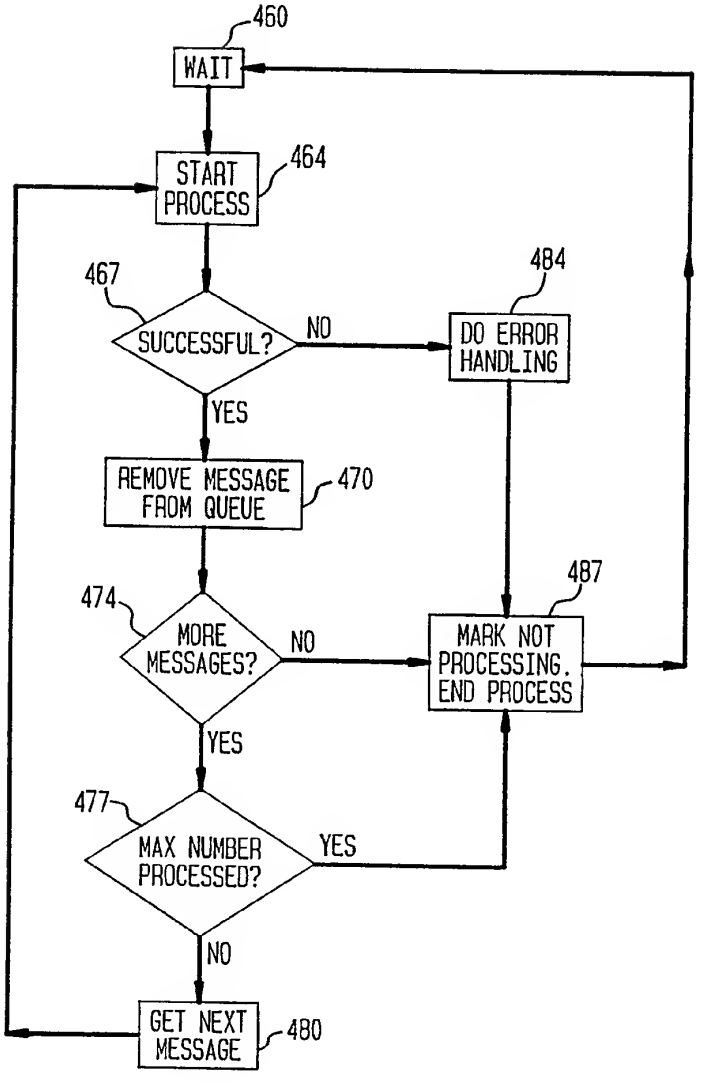
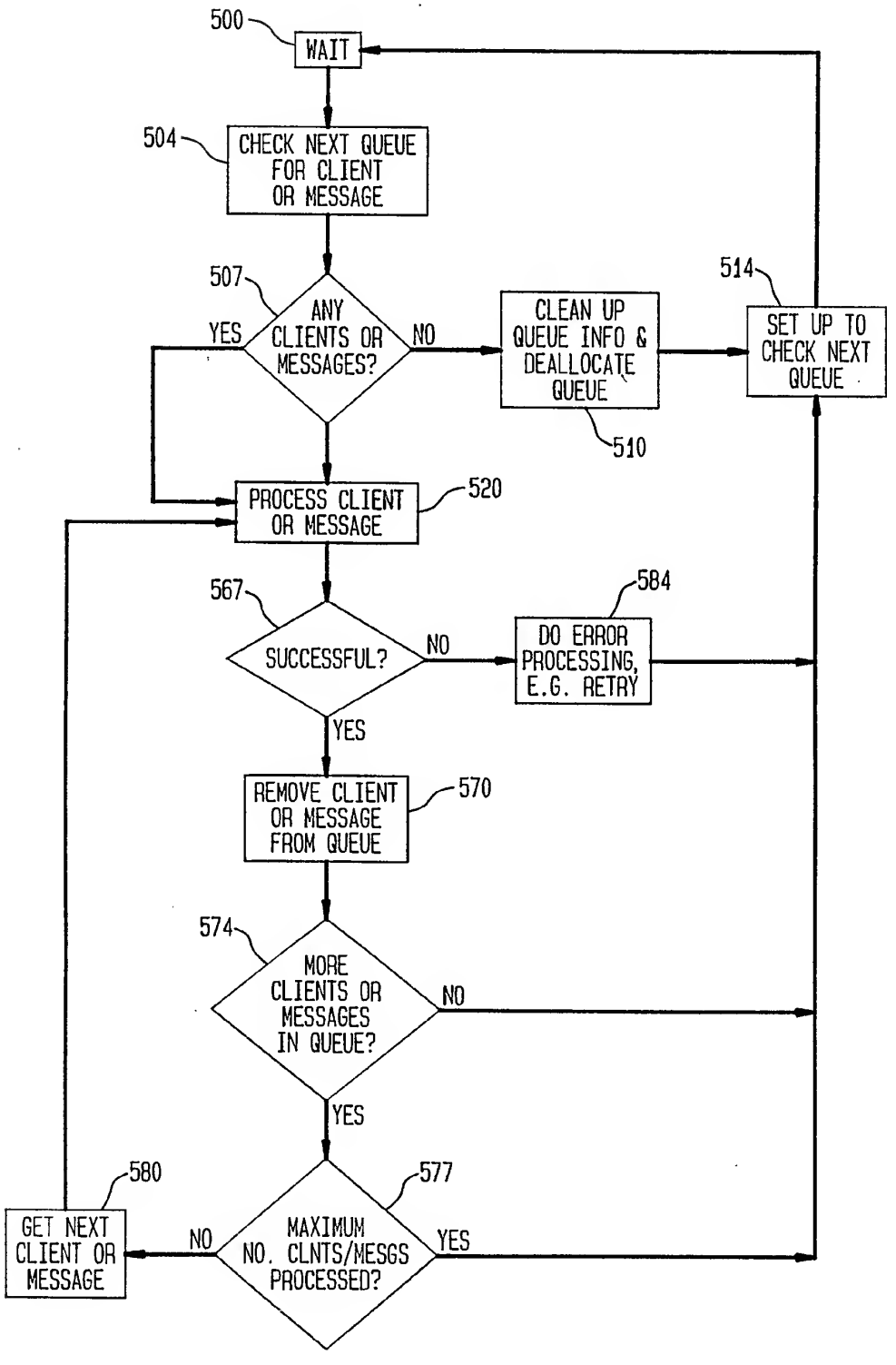

460
WAIT
START PROCESS
464
467
SUCCESSFUL?
NO
484
DO ERROR HANDLING
YES
REMOVE MESSAGE FROM QUEUE
470
474
MORE MESSAGES?
NO
487
MARK NOT PROCESSING. END PROCESS
YES
477
MAX NUMBER PROCESSED?
YES
NO
GET NEXT MESSAGE
480

FIG. 11


500
WAIT
504
CHECK NEXT QUEUE FOR CLIENT OR MESSAGE
507
ANY CLIENTS OR MESSAGES?
YES
NO
CLEAN UP QUEUE INFO & DEALLOCATE QUEUE
510
514
SET UP TO CHECK NEXT QUEUE
520
PROCESS CLIENT OR MESSAGE
567
SUCCESSFUL?
NO
584
DO ERROR PROCESSING, E.G. RETRY
YES
570
REMOVE CLIENT OR MESSAGE FROM QUEUE
574
MORE CLIENTS OR MESSAGES IN QUEUE?
NO
YES
577
MAXIMUM NO. CLNTS/MESGS PROCESSED?
NO
YES
580
GET NEXT CLIENT OR MESSAGE

INTERNATIONAL SEARCH REPORT

Intern al Application No
PCT/US 94/00826

A. CLASSIFICATION OF SUBJECT MATTER
IPC 5 G06F9/46 H04M3/50

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 5 G06F H04M

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	PROCEEDINGS OF THE 6TH SOUTHERN AFRICAN COMPUTER SYMPOSIUM, 2 July 1991 , CALEDON, SA, pages 69 - 82 A.E. KRZESINSKI ET AL.: 'Product Form Solutions for Multiserver Centres with Hierarchical Concurrency Constraints' see page 69, line 1 - page 70, line 33 see page 79, line 12 - page 81, line 13 ---	1,9,17, 25
A	EP,A,0 236 013 (AT&T) 9 September 1987 see abstract see page 1, line 1 - page 3, line 24 ---	1,9,17, 25

-/--

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
30 May 1994	09. 06. 94
Name and mailing address of the ISA European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax (+31-70) 340-3016	Authorized officer Wiltink, J

Form PCT/ISA/210 (second sheet) (July 1992)

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	PROCEEDINGS OF THE 10TH INTERNATIONAL CONFERENCE ON PATTERN RECOGNITION, 16 June 1990 , ATLANTIC CITY, NJ, US, pages 648 - 653 XP166539 C.H. CHIEN ET AL.: 'PARADIGM: An Architecture for Distributed Vision Processing' see page 649, right column, line 11 - line 26 see page 650, left column, line 38 - right column, line 23; figure 1 -----	1,9,17, 25

1

INTERNATIONAL SEARCH REPORT

Information on patent family members

Intern. \al Application No
PCT/US 94/00826

Patent document cited in search report	Publication date	Patent family member(s)		Publication date
EP-A-0236013	09-09-87	US-A-	4757529	12-07-88
		CA-A-	1260583	26-09-89
		JP-A-	62206995	11-09-87